

LSI DOCKET NO. 03-0984

## APPLICATION FOR LETTERS PATENT OF THE UNITED STATES

### CERTIFICATE OF MAILING BY "EXPRESS MAIL"

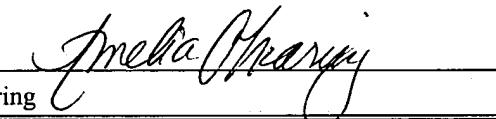
"EXPRESS MAIL" Mailing Label Number EV333421790US

Date of Deposit: October 9, 2003

I HEREBY CERTIFY THAT THIS CORRESPONDENCE, CONSISTING OF 25 PAGES OF SPECIFICATION AND 8 PAGES OF DRAWINGS, IS BEING DEPOSITED WITH THE UNITED STATES POSTAL SERVICE "EXPRESS MAIL POST OFFICE TO ADDRESSEE" SERVICE UNDER 37 CFR 1.10 ON THE DATE INDICATED ABOVE AND IS ADDRESSED TO: MAIL STOP PATENT APPLICATION, COMMISSIONER FOR PATENTS, P.O. BOX 1450, ALEXANDRIA, VA 22313-1450.

BY: \_\_\_\_\_

Amelia C. Nearing



## SPECIFICATION

To all whom it may concern:

Be It Known, That We, **Abhishek Kar**, a citizen of the United States, residing at **2582 West Crawford Street, Tucson, Arizona 85745**, and **Robert Louis Morton**, a citizen of the United States, residing at **11560 East Lusitano Place, Tucson, Arizona 85748**, and **Gary William Steffens**, a citizen of the United States, residing at **3800 North Cherry Creek Place, Tucson, Arizona 85749**, have invented certain new and useful improvements in "**Method, System, and Product for Proxy-Based Method Translations for Multiple Different Firmware Versions**", of which We declare the following to be a full, clear and exact description:

## BACKGROUND OF THE INVENTION

### 1. Technical Field:

The present invention relates generally to data processing systems such as computer systems, and more particularly to proxy-based API translations for multiple different firmware versions.

### 2. Description of the Related Art:

A computer system includes firmware which provides an interface between the computer system's hardware and software. Typically, an operating system interacts with the hardware utilizing the firmware.

The hardware of a computer system is often revised and improved. It is desirable for software applications to interface with different versions of a computer system. When updates to the hardware occur, the firmware is also usually updated. Thus, the software applications that run on this hardware may need to interact with different versions of firmware so that the software application can interact with the different versions of firmware.

Each version of firmware will include a set of application programming interfaces (APIs) which are used by the software for accessing the firmware. One method for permitting software to interact with the firmware API is to provide interfaces, for example remote procedure calls (RPCs), between the software layer and the firmware API layer. The firmware interface includes methods and classes that are used to represent the current configuration of the hardware and to change the configuration. To differentiate the classes from one firmware release to another for a particular type of API, the firmware changes the class name such that a different Java class name exists for each firmware release for that type of API. The interface software must be written so that it will recognize the different class names for each firmware release. To create a separate class for each method call for each release would create a lot of interface code. In addition, most of the interface code for a new release would be duplicated from the previous release. These interfaces each represent a small slice of the total firmware APIs. Therefore, in order to represent all of the firmware APIs, many interfaces are required. For example, there may be hundreds of interfaces

which when taken together represent an entire set of firmware APIs.

These firmware APIs are specific to a particular version of the firmware. Thus, software that was written to access a version of firmware using one set of firmware APIs will not be able to access a different version of firmware which uses a different set of firmware APIs. The interfaces 5 can be thought of as a software layer above the firmware API layer. These interfaces must have information about a specific version of firmware APIs.

The prior art provides for code that recasts an interface call to a particular firmware version API call. This code consists of a collection of different classes that recast the interface to each version of firmware. For each interface, there will be as many classes as there are firmware 10 versions in order to recast the interface for each version. This code is referred to below as an implementation instance. Thus, there is a different implementation class to create each implementation instance for each interface for each version of supported firmware. When there are hundreds of interfaces, hundreds of classes are necessary to permit the interfaces to interact with each possible firmware API version.

15 For example, **Figure 1** is a block diagram, which follows the Unified Modeling Language (UML) notation, that depicts two different interfaces and the classes that are necessary to support these interfaces with different versions of firmware in accordance with the prior art. An interface **A 100** represents a portion A of the firmware. An interface **B 114** represents a portion B of the firmware. Interface **A 100** may access a portion A instance of version 1 **104** of the firmware using 20 implementation A version 1 instance **102**. Thus, when interface **A 100** is used in a system that includes firmware version 1, implementation instance **102** is used to access portion A.

Interface **A 100** may access a portion A instance of version 2 **108** of the firmware using implementation A version 2 instance **106**. Interface **A 100** may access a portion A of version 3 **112** of the firmware using implementation A version 3 instance **110**.

25 Interface **B 114** may access a portion B instance of version 1 **118** of the firmware using implementation B version 1 instance **116**. Interface **B 114** may access a portion B of version 2 **122** of the firmware using implementation B version 2 instance **120**. Interface **B 114** may access a portion B of version 3 **126** of the firmware using implementation B version 3 instance **124**. For example, implementation A version 1 instance **102** implements interface **A 100** and recasts the

LSI DOCKET NO. 03-0984

APIs found in interface A **100** into their equivalent version-specific APIs of firmware A version 1 instance **104**.

Each implementation instance is an instance of a different class. For example, implementation A version 1 instance **102** is an instance of implementation A version 1 class **130**.

5 Implementation A version 2 instance **106** is an instance of implementation A version 2 class **132**. Implementation A version 3 instance **110** is an instance of implementation A version 3 class **134**. Implementation B version 1 instance **116** is an instance of implementation B version 1 class **136**. Implementation B version 2 instance **120** is an instance of implementation B version 2 class **138**. Implementation B version 3 instance **124** is an instance of implementation B version 3 class **140**.

10 Therefore, in order to implement an interface a particular number of firmware versions requires that number of classes. This number is multiplied by the total number of interfaces in order to implement all of the firmware APIs.

In addition, there are other objects that do the work of creating the implementation instance from the implementation classes. There is a factory object to create a factory instance for each 15 firmware version for creating the implementation instances for each version of firmware. **Figure 2** is a UML diagram that illustrates a factory interface that may be used to create factory objects in accordance with the prior art. Factory interface **200** may be used to create factory instance version 1 **202**, factory instance version 2, **204**, and factory instance version 3 **206**. Factory instance version 1 **202** is an instance of factory class version 1 **208**. Factory instance version 2 **204** is an instance of 20 factory class version 2 **210**. And, factory instance version 3 **206** is an instance of factory class version 3 **212**.

Factory instance version 1 **202** may be used to create the implementation instances for each interface for firmware version 1. Factory instance version 1 **202** is used to create implementation A version 1 **102** and implementation B version 1 **116**. Factory instance version 2 **204** is used to create 25 implementation A version 2 **106** and implementation B version 2 **120**. And, factory instance version 3 **206** is used to create implementation A version 3 **110** and implementation B version 3 **124**. Whenever a new firmware version needs to be supported, a new factory instance must be created which will create the implementation instance for each interface for that version.

LSI DOCKET NO. 03-0984

As should be apparent from the description above, many thousands of lines of code must be written to create all of the implementation classes and factory classes that are needed to create the various objects.

Therefore, a need exists for a method and system for reducing the amount of code that is  
5 needed to support multiple different versions of firmware.

**SUMMARY OF THE INVENTION**

The present invention is a method and system for translating method calls to version-specific method calls. An interface to an underlying object is provided. Applications 5 communicating with the underlying object use the interface. The interface is separate from the underlying object. Version-specific underlying objects are generated. Each one of the version-specific underlying objects is a different version of the underlying object. A translation object is generated for communicating between the interface and each one of the version-specific underlying objects. The translation object is used for translating an interface method call 10 invoked on the interface to a version-specific method call for the underlying object for each version of the underlying object. The translation object is generated from a single proxy class and a single InvocationHandler class. The same proxy class and the same InvocationHandler class are used to generate the translation object for each different version of the underlying object.

15 The above as well as additional objectives, features, and advantages of the present invention will become apparent in the following detailed written description.

**BRIEF DESCRIPTION OF THE DRAWINGS**

The novel features believed characteristic of the invention are set forth in the appended claims. The invention itself however, as well as a preferred mode of use, further objects and 5 advantages thereof, will best be understood by reference to the following detailed description of an illustrative embodiment when read in conjunction with the accompanying drawings, wherein:

**Figure 1** is a block diagram, which follows the Unified Modeling Language (UML) convention, that depicts two different interfaces and the classes that are necessary to support these interfaces with different versions of firmware in accordance with the prior art;

10 **Figure 2** is a UML diagram that illustrates a factory interface and the classes that are necessary to create factory objects in accordance with the prior art;

**Figure 3** is a UML diagram that depicts two different interfaces and the classes that are necessary to support these interfaces with different versions of firmware in accordance with the present invention;

15 **Figure 4** is a UML diagram that illustrates a factory interface and the classes that are necessary to create factory objects in accordance with the present invention;

**Figure 5A** depicts a high level flow chart which illustrates the process of a proxy instance in accordance with the present invention;

20 **Figure 5B** illustrates a high level flow chart which depicts the process of an InvocationHandler instance in accordance with the present invention;

**Figure 6** depicts a high level flow chart which illustrates the process of a factory object that creates a proxy instance and an InvocationHandler instance for a particular firmware version in accordance with the present invention; and

25 **Figure 7** illustrates a computer system that may be used to implement the present invention in accordance with the present invention.

**DETAILED DESCRIPTION**

The description of the preferred embodiment of the present invention has been presented for purposes of illustration and description, but is not intended to be exhaustive or limited to the 5 invention in the form disclosed. Many modifications and variations will be apparent to those of ordinary skill in the art. The embodiment was chosen and described in order to best explain the principles of the invention and the practical application to enable others of ordinary skill in the art to understand the invention for various embodiments with various modifications as are suited to the particular use contemplated.

10       Figure 3 is a UML diagram that depicts two different interfaces and the classes that are necessary to support these interfaces with different versions of firmware in accordance with the present invention. Those skilled in the art will recognize that although Figure 3 depicts only two interfaces, any number of interfaces may be implemented in a similar manner. A translation object is depicted which is used to recast method calls from a particular interface to the call 15 required by a particular firmware version. Multiple different proxy instances and invocation handler instances are depicted. Each proxy instance include a reference to, and thus includes, an invocation handler instance. Thus, a translation object that includes a proxy instance and an invocation handler instance are depicted for each version of the firmware, i.e. for each firmware instance. According to an important feature of the present invention, a translation object for each 20 interface is created from a single proxy class and a single invocation handler class.

For example, a first translation object includes proxy instance 302 and invocation handler instance 303. A second translation object includes proxy instance 306 and invocation handler instance 307. A third translation object includes proxy instance 310 and invocation handler instance 311. A fourth translation object includes proxy instance 316 and invocation handler instance 317. A fifth translation object includes proxy instance 320 and invocation handler instance 321. And, a sixth translation object includes proxy instance 324 and invocation handler instance 325.

The invocation handlers are referred to herein as "InvocationHandler".

An interface A 300 represents a portion A of the firmware. Interface A 300 may access a

LSI DOCKET NO. 03-0984

portion A instance of version 1 304 of the firmware using interface proxy instance A1 302 and interface InvocationHandler instance A1 303. Thus, when interface A 300 is used in a system that includes firmware version 1, interface proxy instance A1 302 and interface InvocationHandler instance A1 303 are used to access portion A. When interface A 300 is used in a system that

5 includes firmware version 2, interface proxy instance A2 306 and interface InvocationHandler instance A2 307 are used to access portion A. When interface A 300 is used in a system that includes firmware version 3, interface proxy instance A3 310 and interface InvocationHandler instance A3 311 are used to access portion A. Interface A 300 is a union of all methods available in all versions of the corresponding firmware object.

10 An interface B 314 represents a portion B of the firmware. When interface B 314 is used in a system that includes firmware version 1, interface proxy instance B1 316 and interface InvocationHandler instance B1 317 are used to access portion B. When interface B 314 is used in a system that includes firmware version 2, interface proxy instance B2 320 and interface InvocationHandler instance B2 321 are used to access portion B. When interface B 314 is used in a

15 system that includes firmware version 3, interface proxy instance B3 324 and interface InvocationHandler instance B3 325 are used to access portion B.

The present invention provides for the creation of all proxy instances and all InvocationHandler instances which are necessary for all interfaces utilizing only two classes. Thus, only two classes are necessary in order to provide functionality that is equivalent to the six different

20 implementer classes depicted by **Figure 1** of the prior art. For a system having 290 different interfaces, the prior art required 290 different classes to create the 290 implementation instances. According to the present invention, for a system having 290 different interfaces, only two classes are necessary to create the 290 different translation objects.

Proxy class 330 and interface InvocationHandler class 332 are the only classes that are

25 necessary in order to create the translation object of the an implementer layer between the interface layer and the firmware API layer for any firmware version. Further, the Java programming language includes the Proxy class and an InvocationHandler abstract class. Thus, the present invention describes using this Proxy class with a new class which is a subclass of the InvocationHandler, i.e. the interface InvocationHandler class, in order to create a translator object

that recasts methods invoked on the interface to the equivalent method to be invoked on the particular version of the firmware.

Those skilled in the art will recognize that although the Java programming language is used, other programming languages may instead be used.

5       **Figure 4** is a UML diagram that illustrates a factory interface and the classes that are necessary to create factory objects in accordance with the present invention. Factory proxy instance version 1 including factory InvocationHandler instance 1 **402**, factory proxy instance version 2 including factory InvocationHandler instance 2 **404**, and factory proxy instance version 3 including factory InvocationHandler instance **406** all inherit from factory interface **400**.  
10      Similarly to the description of **Figure 3**, only two classes, in addition to the interface class, are needed in order to create instance **402**, instance **404**, and instance **406**. Proxy class **408**, which is the same as Proxy class **320**, and factory InvocationHandler class **410** are used to create factory proxy instance version 1 including factory InvocationHandler instance 1 **402**, factory proxy instance version 2 including factory InvocationHandler instance 2 **404**, and factory proxy instance  
15      version 3 including factory InvocationHandler instance 3 **406**.

20       **Figure 5A** depicts a high level flow chart which illustrates the process of a proxy instance in accordance with the present invention. The process starts as depicted by block **500** and thereafter passes to block **502** which illustrates a method of a particular interface being invoked by a client on a proxy instance. Next, block **503** depicts the proxy instance passing the method to its InvocationHandler instance. The process then passes to block **504** shown in **Figure 5B**.

25       **Figure 5B** illustrates a high level flow chart which depicts the process of an InvocationHandler in accordance with the present invention. Block **504** depicts a determination of whether or not any of the method's parameters are proxy instances. If a determination is made that at least one of the method's parameters are a proxy instance, the process passes to block **505** which illustrates the parameters now matching the parameter format for the firmware API that is to be invoked. Next, block **506** illustrates replacing each actual parameter that is a proxy instance with its associated underlying firmware API instance. Next, block **508** depicts replacing each formal parameter that is a proxy instance with its underlying firmware API object type. The process then passes to block **505**.

Block 510, then, illustrates the InvocationHandler instance finding the firmware API method to call based on the interface method's name and the API method's formal parameter types. This involves the ability, during runtime, of looking up a class, and finding the methods defined in the class which includes information about formal parameters and return values. Knowing the 5 method name, its formal parameter types, and the actual parameters enables dynamic invocation of the method. Thereafter, block 512 depicts the InvocationHandler instance using Java reflection to invoke the firmware API method passing it the actual parameters. The process then passes to block 514 which illustrates a determination of whether or not the interface method's return type is an interface. If a determination is made that the interface method's return type is not an interface, the 10 process passes to block 516 which depicts the InvocationHandler instance returning the firmware API method's return object as the return value to the client.

Referring again to block 514, if a determination is made that the interface method's return type is an interface, the process passes to block 518 which illustrates a determination of whether or not the interface method's return type is an array. If a determination is made that the interface 15 method's return type is not an array, the process passes to block 520 which depicts the InvocationHandler instance creating a proxy instance that implements the interface return type for the object returned from the firmware API method call. The process then passes to block 522 which illustrates the InvocationHandler instance returning the single proxy instance as the return value to the client. Referring again to block 518, if a determination is made that the interface 20 method's return type is an array, the process passes to block 524 which illustrates creating a proxy instance that implements the interface return type for each object in the array returned from the firmware API method call. The new proxy instances are put into a new array. Next, block 526 depicts returning the array of proxy instances as the return value to the client.

The process depicted by blocks 514-526 is executed by a factory proxy instance.

25 **Figure 6** depicts a high level flow chart which illustrates the process of a factory instance that creates an interface proxy instance including an interface InvocationHandler instance for a particular firmware version in accordance with the present invention. The process starts as depicted by block 600 and thereafter passes to block 602 which illustrates discovering a storage array. Next, block 604 depicts determining the current version of the firmware. Block 606, then, depicts

determining whether a factory proxy instance currently exists for the determined version of firmware. A factory object may be thought of as a building object that is used to create an interface proxy instance and an interface InvocationHandler instance for a specific firmware version. If a determination is made that a factory proxy instance does not exist for the current firmware version,  
5 the process passes to block **608** which illustrates creating a new factory proxy instance for the specified firmware version. The process then passes to block **610**. Referring again to block **606**, if a determination is made that a factory proxy instance does exist for the specified firmware version, the process passes to block **610**.

Block **610**, then, depicts the factory proxy instance using Java reflection to create an  
10 interface InvocationHandler instance for this firmware version. Next, block **612** illustrates the factory proxy instance creating an interface proxy instance that implements the specified interface and includes a reference to the newly created interface InvocationHandler instance. Thus, the interface proxy instance may be thought of as including the interface InvocationHandler instance. The process then passes to block **614** which depicts the factory proxy instance returning the  
15 interface proxy instance which includes the interface InvocationHandler instance.

**Figure 7** is an illustration of a computer system that may be used to implement the present invention in accordance with the present invention. Data processing system **700** may be a symmetric multiprocessor (SMP) system including a plurality of processors **702** and **704** connected to system bus **706**. Alternatively, a single processor system may be employed. Also connected to system bus **706** is memory controller/cache **708**, which provides an interface to local memory **709**.  
20 I/O bus bridge **710** is connected to system bus **706** and provides an interface to I/O bus **712**. Memory controller/cache **708** and I/O bus bridge **710** may be integrated as depicted.

Peripheral component interconnect (PCI) bus bridge **714** connected to I/O bus **712** provides an interface to PCI local bus **716**. A number of modems may be connected to PCI bus **716**. Typical  
25 PCI bus implementations will support four PCI expansion slots or add-in connectors. Communications links to network computers may be provided through modem **718** and network adapter **720** connected to PCI local bus **716** through add-in boards.

Additional PCI bus bridges **722** and **724** provide interfaces for additional PCI buses **726** and **728**, from which additional modems or network adapters may be supported. In this manner, data

processing system 700 allows connections to multiple network computers. A memory-mapped graphics adapter 730 and hard disk 732 may also be connected to I/O bus 712 as depicted, either directly or indirectly.

Those of ordinary skill in the art will appreciate that the hardware depicted in **Figure 7** may vary. For example, other peripheral devices, such as optical disk drives and the like, also may be used in addition to or in place of the hardware depicted. The depicted example is not meant to imply architectural limitations with respect to the present invention.

The following is an example of the present invention:

(This is the interface.)

10 StorageInterface exposes method to get StorageArray objects (disk drive, volume etc)

```
interface StorageInterface {  
    GetDiskDrive(...);  
    GetStorageVolume(...);  
    ...  
15 }
```

(This is the InvocationHandler instance.)

```
class StorageHandler {  
    invoke(...){  
20    }  
    } //call this instance Handler1
```

When a particular translation object instance, referred to below as translation1, is created for StorageInterface, we create an instance of Java's Proxy class and designate the StorageInterface as 25 the external view of this instance of the Proxy class. We create an instance of the StorageHandler, i.e. Handler1, and set that as the InvocationHandler instance included within this translation object.

So we get a new instance that looks like:

```
StorageInterface{ //anybody referring to this instance will see what looks like a  
    //StorageInterface
```

```
Underneath is Java's Proxy object {  
    Contains Handler1 which is an instance of the InvocationHandler  
}  
} //Call this instance translation1
```

5

When the client calls getDiskDrive() on this new instance "translation1", the call is handled by the InvocationHandler of the proxy instance (i.e. in response to translation1.getDiskDrive()...the Proxy (Java) implementation will call the invoke method of Handler1 and pass in arguments that specify what the original call to translation1 was. The InvocationHandler 10 Handler1 will respond as required and return a the requested object.

It is important to note that while the present invention has been described in the context of a fully functioning data processing system, those of ordinary skill in the art will appreciate that the processes of the present invention are capable of being distributed in the form of a computer readable medium of instructions and a variety of forms and that the present invention applies 15 equally regardless of the particular type of signal bearing media actually used to carry out the distribution. Examples of computer readable media include recordable-type media such a floppy disc, a hard disk drive, a RAM, and CD-ROMs and transmission-type media such as digital and analog communications links.

The description of the present invention has been presented for purposes of illustration 20 and description, and is not intended to be exhaustive or limited to the invention in the form disclosed. Many modifications and variations will be apparent to those of ordinary skill in the art. The embodiment was chosen and described in order to best explain the principles of the invention, the practical application, and to enable others of ordinary skill in the art to understand the invention for various embodiments with various modifications as are suited to the particular 25 use contemplated.